UNIVERSITY OF
ILLINOIS LIBRARY
AT URBANA-CHAMPAIGN

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

To renew call Telephone Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

L161—O-1096

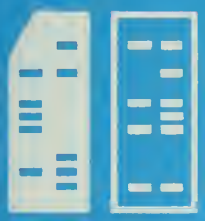
UIUCDCS-R-76-809
Math
no.809
Cop 2

MUMS: A MODULAR UNIFIED MICROPROCESSOR SYSTEM

by

Robert Walter Catlin

June 1976



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN URBANA, ILLINOIS

The Library of the
SEP 14 1976
University of Illinois

Digitized by the Internet Archive
in 2013

http://archive.org/details/mumsmodularunifi809catl

UIUCDCS-R-76-809

MUMS: A MODULAR UNIFIED MICROPROCESSOR SYSTEM

by

Robert Walter Catlin

June 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

ACKNOWLEDGMENT

The author wishes to thank his thesis advisor, Professor Michael Faiman, for his help and friendship.

Also, he would like to thank the other graduate students of the Information Engineering Laboratory for their advice and help, and Miss Pamela Farr for the typing of this thesis.

TABLE OF CONTENTS

		PAGE
1.	INTRODUCTION	1
	1.1 Applications of Microcomputers	1
	1.2 Common Properties of Microcomputers	3
	1.3 Design Flexibility	4
2.	DETAILED BUS DESIGN AND RATIONALE	6
	2.1 Memory Interface and Timing	6
	2.2 I/O Interface and Timing	8
	2.3 Interrupt Scheme and Timing	10
	2.4 DMA Scheme and Timing	16
	2.5 Bus Modularity	19
	2.6 Layout of Bus Signals	19
	2.7 Bus Termination	22
3.	MICROCOMPUTER INTERFACES	24
	3.1 Intel 8080 MUMS CPU Module	24
	3.2 Other CPU Interfaces	28
4.	OTHER SYSTEM MODULES	30
	4.1 4K x 8 RAM Module	30
	4.2 Read-Only Memory Module	32
	4.3 Serial I/O Interface Module	32
	4.4 Shared Memory Display Controller	37
	4.5 Control/Display Module	42
5.	MUMS SYSTEM SOFTWARE	43
	5.1 8080 Processor Initialization	43
	5.2 Universal DEBUG Monitor and Cross-Assembler	45
6.	CONCLUSION	46
REFERENCES		47
APPENDIX A		48

LIST OF FIGURES

FIGURE		PAGE
1	Typical read and write cycles	9
2	Typical input and output cycles	11
3	Interrupt request timing	13
4	Interrupt device logic	15
5	DMA timing	17
6	DMA request logic	18
7	Backplane signal layout (pin side view)	20
8	Termination network	23
9	Intel 8080 MUMS CPU interface	25
10	MUMS 4K x 8 RAM module	31
11	Read-only memory module	33
12	Serial I/O interface module	35
13	Display Controller (Character Generation)	39
14	Display Controller (Refresh Address Generation)	40
15	Display Controller (Refresh Memory)	41

1. INTRODUCTION

1.1 Applications of Microcomputers

Microcomputers have spawned a new direction in Computer Science. With the availability of a computer that is both small and very inexpensive, hundreds of applications now appear that can benefit from the versatility and simplicity of software implemented logic. Such applications as small, dedicated machine controllers were, in the past, not cost effective when considering the use of a minicomputer. Even the automotive industry now considers the use of a computer for ignition timing and fuel injection control in every automobile.

Microcomputers are not a different species of computer. They are only smaller, cheaper and, at this point in time, slower than computers built of other technologies. Since they have the same basic organization as larger machines, and are orders of magnitude less expensive, microcomputers could make an ideal tool for the teaching of Computer Science. Machine language programming, simple assembly language and system programming, and interface logic design could all be taught effectively using a microcomputer system. Actually letting a class interface their experimental logic to an expensive and usually busy minicomputer system could never be allowed. However, a student could get "hands-on" experience with an expendable computer. A modular microcomputer system can be designed in such a way as to allow a student to choose only those functional modules that would be needed to perform a given experiment. These hardware and software modules could all be taken off-the-shelf, or individual modules could be designed and implemented as a part of the student's experiment. In this way, a student does not have to design and build an entire system

from scratch only to experiment with a single aspect of the system. The student implemented module could be anything from Random Access or Read-Only memory, to a new microcomputer or I/O interface and the software to support it. The student could even be introduced to multiprocessing through the use of two or more such modular systems.

This Modular Unified Microcomputer System (MUMS) could also be very useful as a laboratory research tool. An increasing number of thesis projects employ microcomputers in their structure. It should not be the researcher's task to duplicate the development of those portions of a microcomputer system that have already been built and debugged by others. Certain modules are likely to be needed in most systems. These include RAM, ROM, serial I/O, mass storage interfaces, and of course, a CPU module. If a set of such universal modules were available off-the-shelf, much duplication of effort would be saved and a researcher could better direct his talents toward less mundane tasks. A modular system would also allow experimental systems to be reconfigured rapidly to explore different architectural solutions to a given problem. An inexpensive tool for investigating hardware/software tradeoffs in multiprocessing systems would be realized in this manner.

MUMS could also be used as a developmental tool for production systems. Different processing elements could be tried driving the same special purpose interfaces that comprise the end item's function. Comparisons can then be made more easily with regard to speed, cost, and efficiency of code between different processors.

1.2 Common Properties of Microcomputers

Almost all computers, including microcomputers, have certain properties in common. The fact that all machines must reference memory to manipulate instructions and data, and communicate to the outside world through peripheral devices is very important. To design a modular system that can support a variety of processing elements, the common properties must first be isolated. A set of generic control signals can then be defined to create a common bus. This has been done by minicomputer manufacturers on such machines as the DEC PDP-11 and Lockheed SUE. Bus protocols like the UNI-BUS and INFI-BUS however, would be overkill for a microcomputer based system. Another modular digital system has been defined primarily as an input/output subsystem. The CAMAC modular instrumentation standard [1] was meant to provide a "crate" of cards, separated from a supporting processor. The CAMAC bus is a digital dataway that connects the I/O cards to the control module, which in turn, interfaces to the host processor. This bus protocol, as well as those of DEC and Lockheed, is too complicated for use in a system that is to be used as an educational tool. These standards provide capabilities that were not deemed necessary and merely added to the cost of a system.

Microcomputers produced to this point all need a certain amount of peripheral circuitry to construct or manipulate control signals from the CPU to the rest of the system. Although every microcomputer has its own method of signalling memory read, write, input and ouptut operations, only a few events are of interest to a generalized memory or I/O device. These generic events can be extracted from any processor with, usually, little more logic than would be necessary to interface a processor to

the system using its own idiosyncratic control signals. The amount of support logic does not increase appreciably just because the control signals are made to conform to some standardization. If these standardized control and data signals are defined carefully, few limitations, if any, will be imposed on any present or foreseeable microcomputer.

1.3 Design Flexibility

There have been two previous microcomputer systems built (posthumously dubbed MUMS I and II) that have proven to be quite flexible in the types of devices that they can support. Incidentally, the only major changes that have been made between MUMS I and MUMS III (described herein) are the connectors used on the backplane and the more powerful interrupt scheme. MUMS II has supported two different processors, a shared memory alphanumeric CRT controller, and a direct memory access vector interpolator used to drive an X-Y CRT display. Of course, more conventional parallel and serial interfaces routinely reside on MUMS II. This flexibility already demonstrated will be enhanced on MUMS III due to the expanded interrupt handling capability and in the manner that spare bus pins have been laid out. It is expected that many more 16 BIT microcomputers will be seen in the near future. The MUMS III bus has been designed with 16 address, 16 data lines and 4 uncommitted lines next to them. It will thus be possible to expand either the addressability or word length of the bus. Other possible uses of the uncommitted address/data lines include multiprocessor addressing when expensive peripherals are to be shared.

It is believed that the generic control signals described in the next section comprise a complete set of functions for a small scale general purpose computer. It may well be that some microcomputer will be introduced with a radically different architecture, and therefore will not fit well into this scheme. However, the vast majority of processors will be able to easily produce the generic MUMS control signals. The utility of the MUMS system has been the primary guideline in its design.

2. DETAILED BUS DESIGN AND RATIONALE

2.1 Memory Interface and Timing

A random access memory module interface must perform only a few simple tasks. It must recognize when the address on the bus is valid and refers to that portion of the address space in which the module resides. The module must only take control of the bus data lines when it is addressed during a read cycle. Data is written into the particular module when it is addressed during a write cycle and the data on the bus is known to be valid. And finally, the module must indicate when data to be read is valid on the bus. These functions can be accomplished in an asynchronous manner using five signals.

$\overline{\text{RAV}}$ - Read Address Valid

This active low signal from the processor to the memory indicates the fact that a read cycle is in progress and that the address on the bus is valid. Knowing that the address is valid, all memory modules decode the upper bits to determine which one should respond. The $\overline{\text{RAV}}$ signal must go inactive between successive read cycles so that memory systems, such as core, can determine the beginning of a cycle and initiate a read-restore operation. Although semiconductor memory systems don't require this precise action, access times must be measured by the interface to signal to the processor that accessed data is valid. Therefore, this control signal must still go inactive between memory cycles to start these timers upon receipt of the next leading edge. As long as $\overline{\text{RAV}}$ is active, the addressed module may control the data lines.

$\overline{WAV}$ - Write Address Valid

This active low signal from the processor to memory indicates that a write cycle is in progress and that the address on the bus is valid. This signal must also go inactive between successive cycles for the same reasons as the $\overline{RAV}$ signal. $\overline{WAV}$ is also active throughout the cycle.

$\overline{WLD}$ - Write Low Data Valid

This active low signal from the processor to memory indicates that data to be written is valid on the lower eight bits of the data bus. The fact that there is a separate signal for the upper and lower eight bits of the bus is for those 16 bit processors that can perform byte operations on memory.

$\overline{WHD}$ - Write High Data Valid

This active low signal from the processor to memory indicates that data is valid on the upper (high order) data bus lines. The use of this signal is otherwise the same as for $\overline{WLD}$. Both $\overline{WLD}$ and $\overline{WHD}$ are asserted for a 16-bit write cycle.

$\overline{ACK}$ - Acknowledge

This active low signal from the addressed memory module to the processor indicates that the module is in the process of responding to a $\overline{RAV}$ or $\overline{WAV}$ signal. The module responds with $\overline{ACK}$ when its address is recognized on the bus. $\overline{ACK}$ is removed when the memory has completed either accessing or writing data. Using this signal makes it possible to have memory modules of varying speeds in a given system.

Figure 1 shows the timing of a typical read cycle and write cycle. Note that there should be some deskew time allowed from when an address is actually made valid to the time $\overline{RAV}$ or $\overline{WAV}$ is asserted.

2.2 I/O Interface and Timing

Input and output operations in the MUMS system were originally to be accomplished using a memory mapping technique only. That is, an I/O device would decode the bus address lines and respond to the $\overline{RAV}$ and $\overline{WAV}$ signals as if the device were a memory location. Memory mapped I/O is a very general scheme that any microcomputer can manage. However, there are two things wrong with a purely memory mapped scheme. First, all input and output modules would have to decode the complete address space. A full 16 bits of decode logic would be required for every I/O module. Secondly, the input and output instructions performed by most microcomputers would not be usable in a purely memory mapped system.

To resolve the first problem, the number of possible I/O ports has been limited to 256. The high order 8 address bits are not decoded by the I/O module. Instead, a particular page of the address space (one of the 256 combinations of the high order 8 address bits) is decoded on the CPU module. The I/O page equality signal ($\overline{I/O}$) is driven down the bus by an open collector gate. I/O modules must only decode the lower 8 bits of the address bus and respond to memory control signals when $\overline{I/O}$ is active. Since the I/O page is decoded on the CPU module, none of the I/O modules need know which memory page it is. Of course, it is necessary that no memory module reside in the address space reserved for the I/O page.

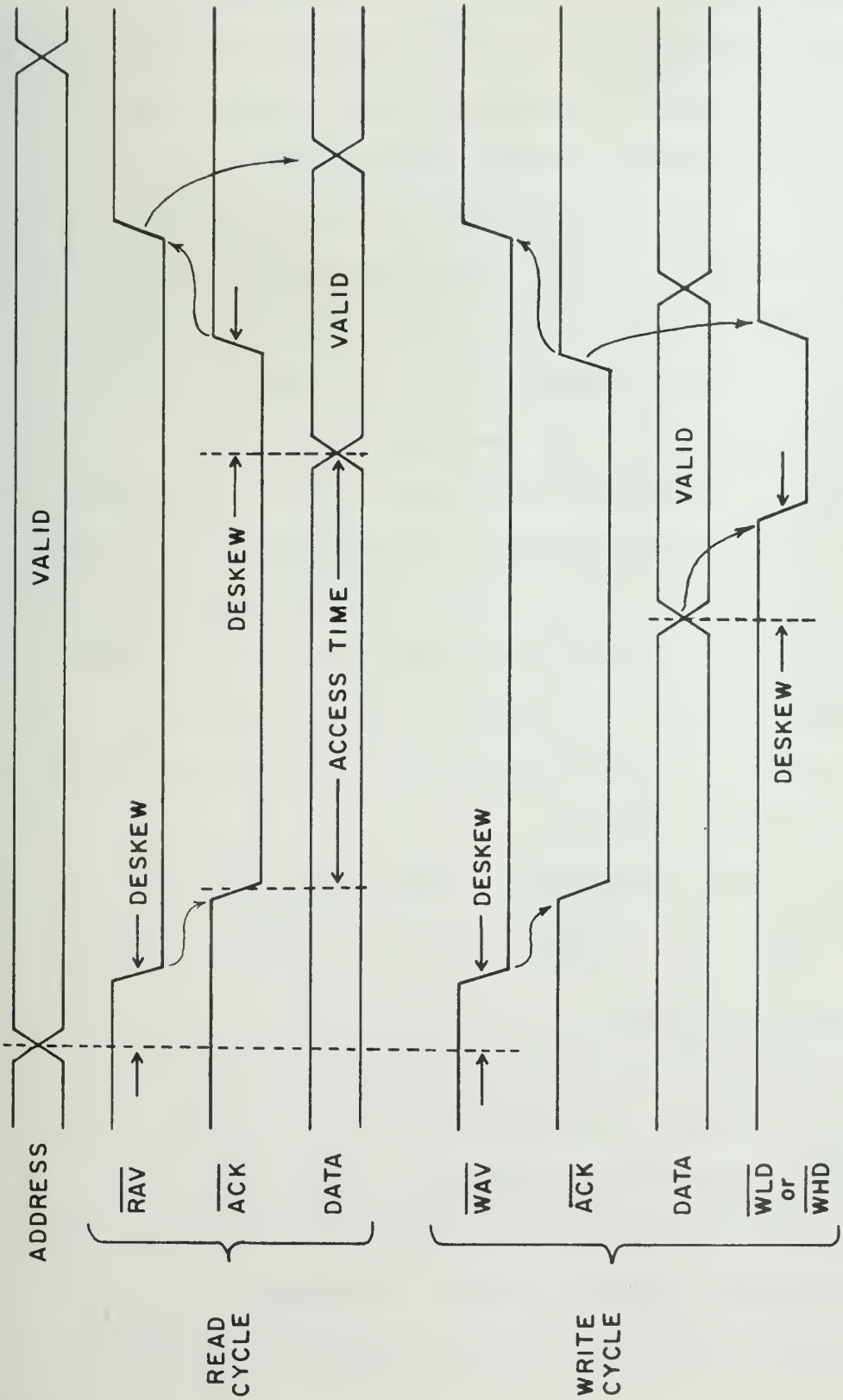


Figure 1. Typical read and write cycles.

The second problem can be resolved on most microcomputers. A typical microcomputer provides both the address of the I/O port being referenced, and control signals to indicate an input or output instruction is being executed. I/O is usually performed through a CPU register. When it is, the address lines on the bus can be used to carry the I/O port address, and the particular control signals provided can be used to generate $\overline{RAV}$, $\overline{WAV}$, $\overline{WLD}$, and $\overline{I/O}$ as required. This is one reason these bus signals are open collector and active low. A bus control signal can be asserted by various portions of the same (or different) module.

There is at least one common microcomputer (the RCA COSMAC) that performs its input and output instructions in a manner that will not allow their use in the MUMS system. There are separate memory address and I/O port address lines provided. The CPU only directs the transfer of data between memory and I/O devices. This doesn't fit into a scheme where there is only one address on the bus at any time. Only memory mapped I/O will be used with such a CPU module.

The $\overline{I/O}$ signal should be asserted sometime before $\overline{RAV}$ or $\overline{WAV}$. The timing for a typical input and output operation is shown in Figure 2.

2.3 Interrupt Scheme and Timing

Most microcomputers have a very simple single level interrupt capability. If more than one interrupt device is to be interfaced to such a system, a polling scheme must be implemented in either software or hardware. Also, some form of priority arbitration must be included in the scheme to handle concurrent interrupt requests. Since there is usually a considerable overhead in entering an interrupt sequence, the polling and

Figure 2. Typical input and output cycles.

priority arbitration should be largely implemented in hardware. It is also desirable to have more than just a single interrupt level so that different classes of interrupting devices can be masked out.

It was mentioned previously that the interrupt schemes implemented in MUMS I and MUMS II were somewhat less powerful than the MUMS III scheme. In particular, the previous schemes only recognized a single interrupt level in hardware and were designed around the INTEL 8080's limited vectoring capability. It was soon realized that this increased the complexity of the interrupt servicing software. If more than seven interrupt devices were interfaced to the bus, polling had to be done in software using specially constructed input ports as flags. Also, there was no way to disable individual interrupting devices.

The MUMS III interrupt scheme provides automatic identification of up to 256 different devices and the capability of disabling any one of them. Three levels of interrupts are recognized and may be disabled from low to high order. Thus, slower or less time critical devices can be blocked from interrupting the servicing of high priority devices.

Three interrupt request ($\overline{IR}_1$, $\overline{IR}_2$, $\overline{IR}_3$), and three interrupt acknowledge lines (IA_1, IA_2, IA_3) correspond to the low to high order interrupt levels. Upon receipt of an interrupt synchronization signal (ISN), a device may assert the $\overline{IR}_n$ signal for its own level. Figure 3 shows the timing of these signals. A device knows that its request is being granted when first, the IA_n signal for its level goes active, and then an interrupt grant (IGI) is received. This grant signal is propagated from the CPU module to the requesting device through a "daisy chain." Each interrupt device must pass the daisy chain signal on to the

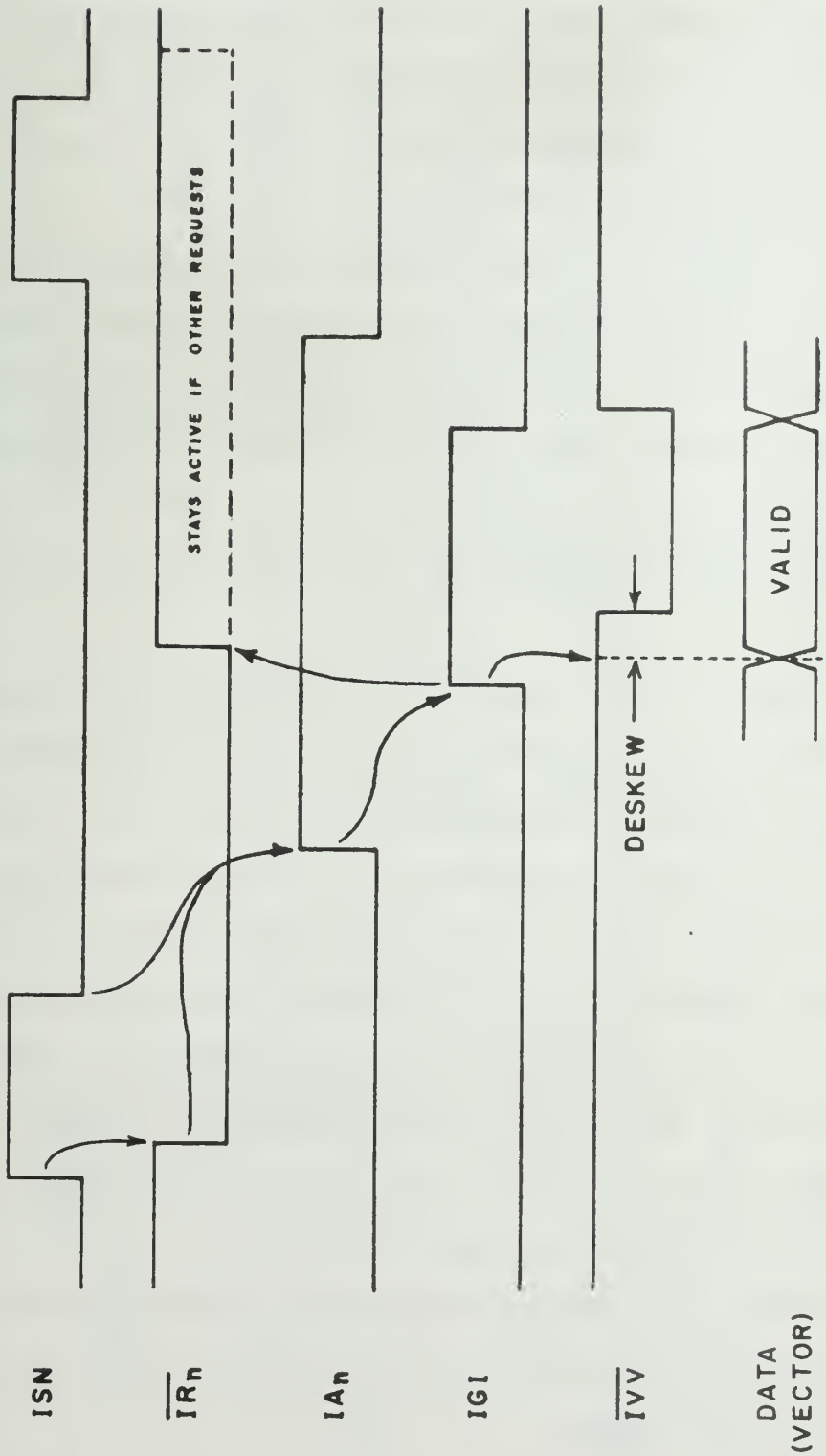


Figure 3. Interrupt request timing.

next module unless the device has requested an interrupt and has received the IA signal for the correct level. Under that set of circumstances, the grant signal (IGO) will not be propagated to the next module, and an identification vector will be gated onto the low order eight data lines by the device. The vector is held on the data lines as long as IGI is active. To indicate that the vector is valid, the responding device asserts $\overline{IVV}$ (Interrupt Vector Valid). The processor can then remove the interrupt grant signal after having recognized the vector. Depending upon the specific implementation, the processor can either do a hardware or software jump to the service routine pointed to by the vector. If in software, the vector can be used as an index into a table of addresses that map physical devices to logical units.

Figure 4 shows a circuit that might be employed with an interrupting device to interface to this protocol. The $\overline{IR}_n$ line will be asserted if the device needs servicing (and its internal enable flip-flop is set) during the leading edge of ISN. Although many devices may interrupt simultaneously, only the one on the highest level, closest to the CPU module will get the daisy-chained grant signal. The IA_n signal being active (high) will allow the lower flip-flop to be clocked when (and if) the grant signal appears at the requesting module. The vector will then be gated onto the data lines (D_0 - D_7) and daisy chain propagation blocked when IGI comes. Note that before either IGI or IA_n go active, both the Set and Reset inputs of the lower flip-flop are active. However, this does not present a problem. Even though Q and $\overline{Q}$ are both high, only one of these outputs is used to provide the propagation or vector enable.

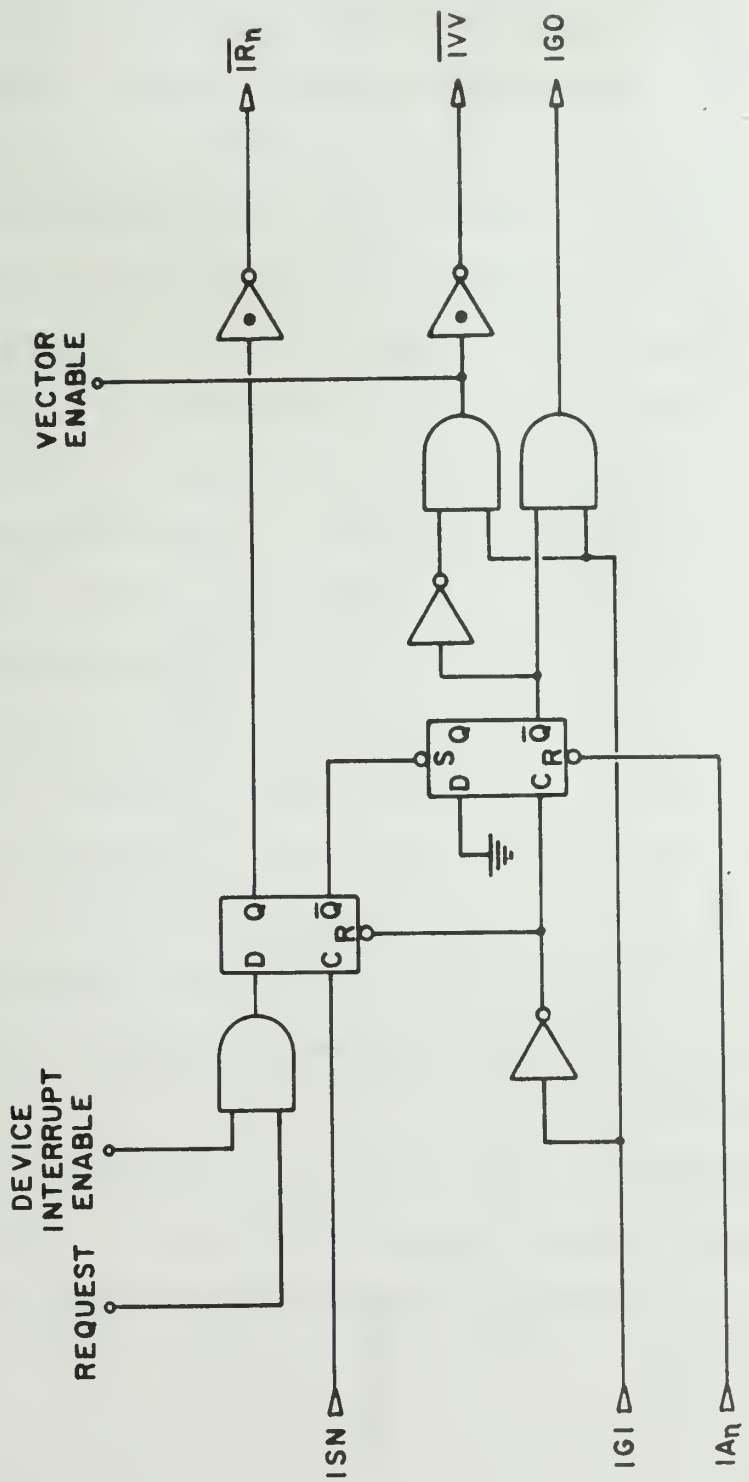


Figure 4. Interrupt device logic.

2.4 DMA Scheme and Timing

Direct Memory Access is very straightforward in most microcomputers as it is in MUMS. A single request line ($\overline{DMR}$) can be asserted by any module, as long as $\overline{DMR}$ is inactive when the request is made. After the processor recognizes the request, a daisy chained DMA grant signal (DGI/DGO) is propagated down the bus. The requesting device that is nearest the CPU module receives the signal which indicates that the CPU has floated the control, address, and data lines. The requesting device can then control memory and I/O as if it were the CPU. It is intended that only one memory or I/O cycle be performed per DMA request. This will make interface of a CPU which employs cycle stealing DMA much easier. Some microcomputers like the INTEL 8080 do have the capability of indefinitely sustaining a DMA transfer as long as the $\overline{DMR}$ line is active. But, any DMA device designed to make use of the feature will not be supported by all microcomputers.

The order of events for a DMA cycle is shown in Figure 5. Control of the memory and I/O signals as well as the data and address busses is the same as described in sections 2.1 and 2.2 except the CPU function is taken over by the DMA device. $\overline{DMR}$ must be held active until the requested cycle is over. New DMA requests should not be made until the grant line has returned to the inactive state.

Figure 6 shows how DMA request logic might be implemented. The upper flip-flop is used to latch a request, and the lower one is used to assure that no short grant pulses are propagated when disengaging from a DMA cycle.

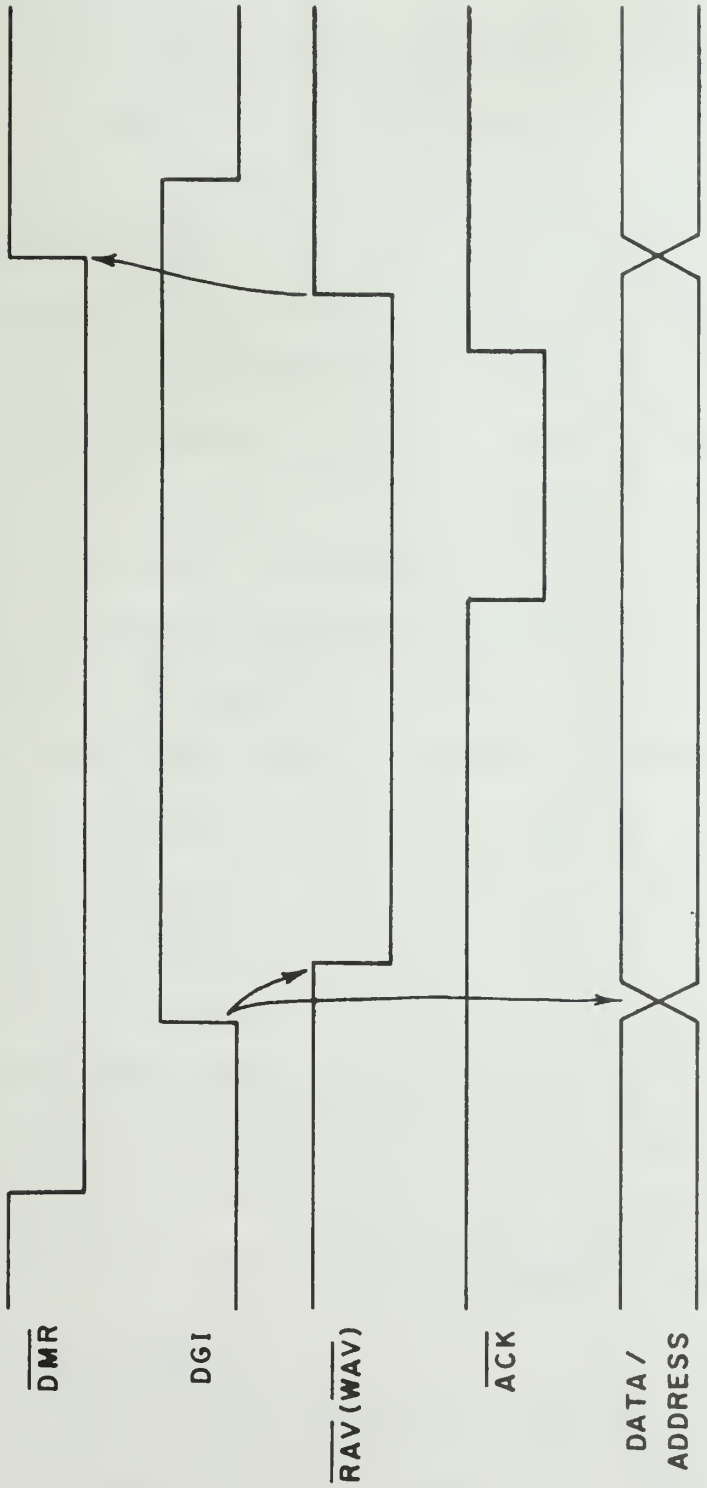


Figure 5. DMA timing.

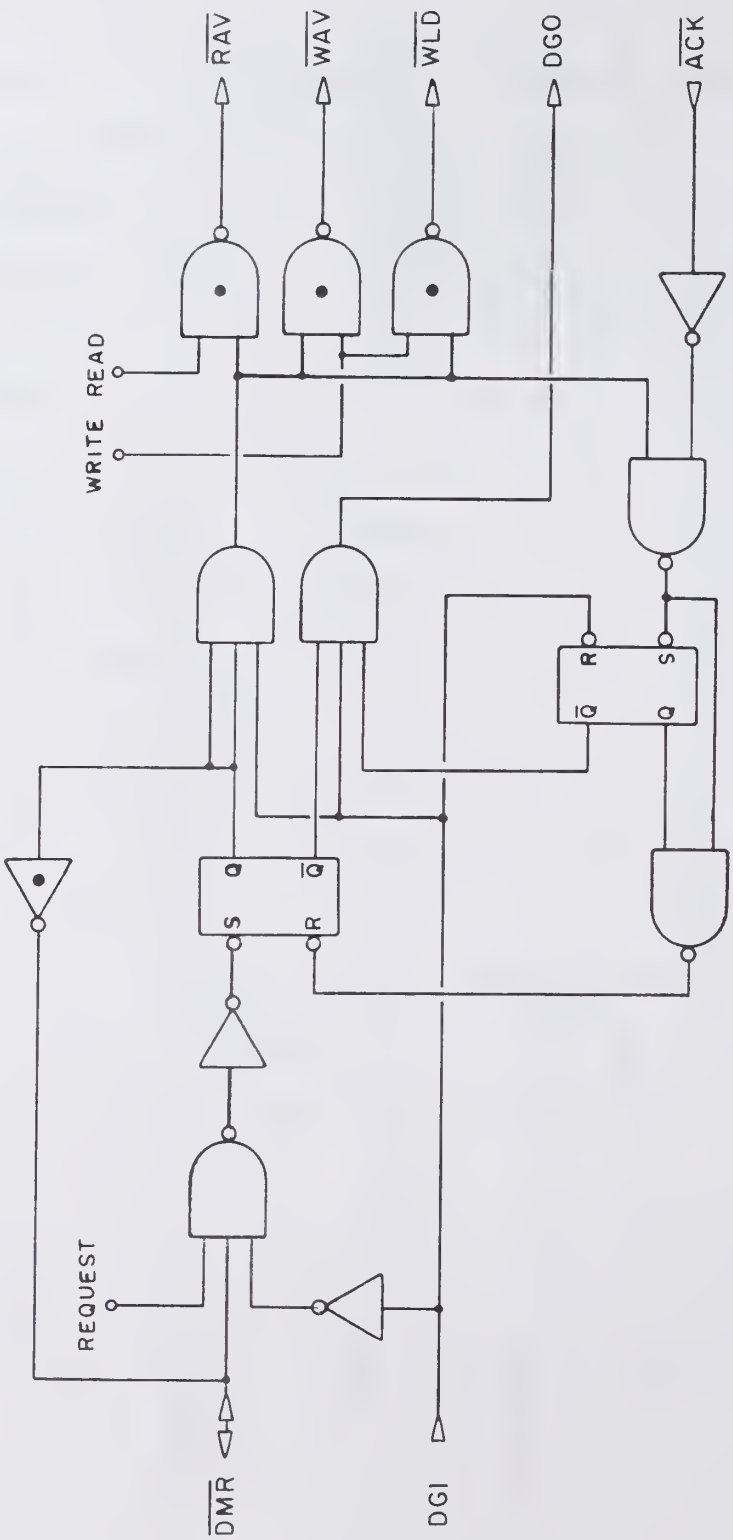


Figure 6. DMA request logic.

2.5 Bus Modularity

Prior experience in building both MUMS I and II has influenced the physical design of MUMS III in a couple of aspects. It was found that the area of a circuit card had to be fairly large to allow a functional module to fit on one card. Also, the card edge connector should have contacts that are spaced at least .125 inches from each other. MUMS II was built on 4.5" X 6" cards with 80 pin connectors (.1" centers). After a fairly small number of insertions, the reliability of the connections was greatly reduced. The tolerance with which .1 inch center connection pads must be etched is also too great for good reliability. For these reasons and the availability of commercial prototyping cards, a DEC "dual card" format was chosen for MUMS III. A "dual card" is 5.25" X 8.5" with 72 contacts on .125 inch centers. The contact fingers are somewhat longer than most. They insert far enough into a DEC H803 connector block to hold the card securely in place without card guides. This particular connector block has four card slots that can be easily interconnected using a backplane circuit card. The MUMS bus may be extended in increments of four slots up to a maximum of 32 slots using these backplane circuit cards. They are designed so that they may be slipped over the pins of a connector block and soldered in place. The backplane card is one slot longer than a single connector block so that it jumps over to the next block and continues the bus.

2.6 Layout of Bus Signals

The actual assignment of MUMS bus signals is shown in Figure 7. The view is from the pin side of a connector block and shows a single slot.

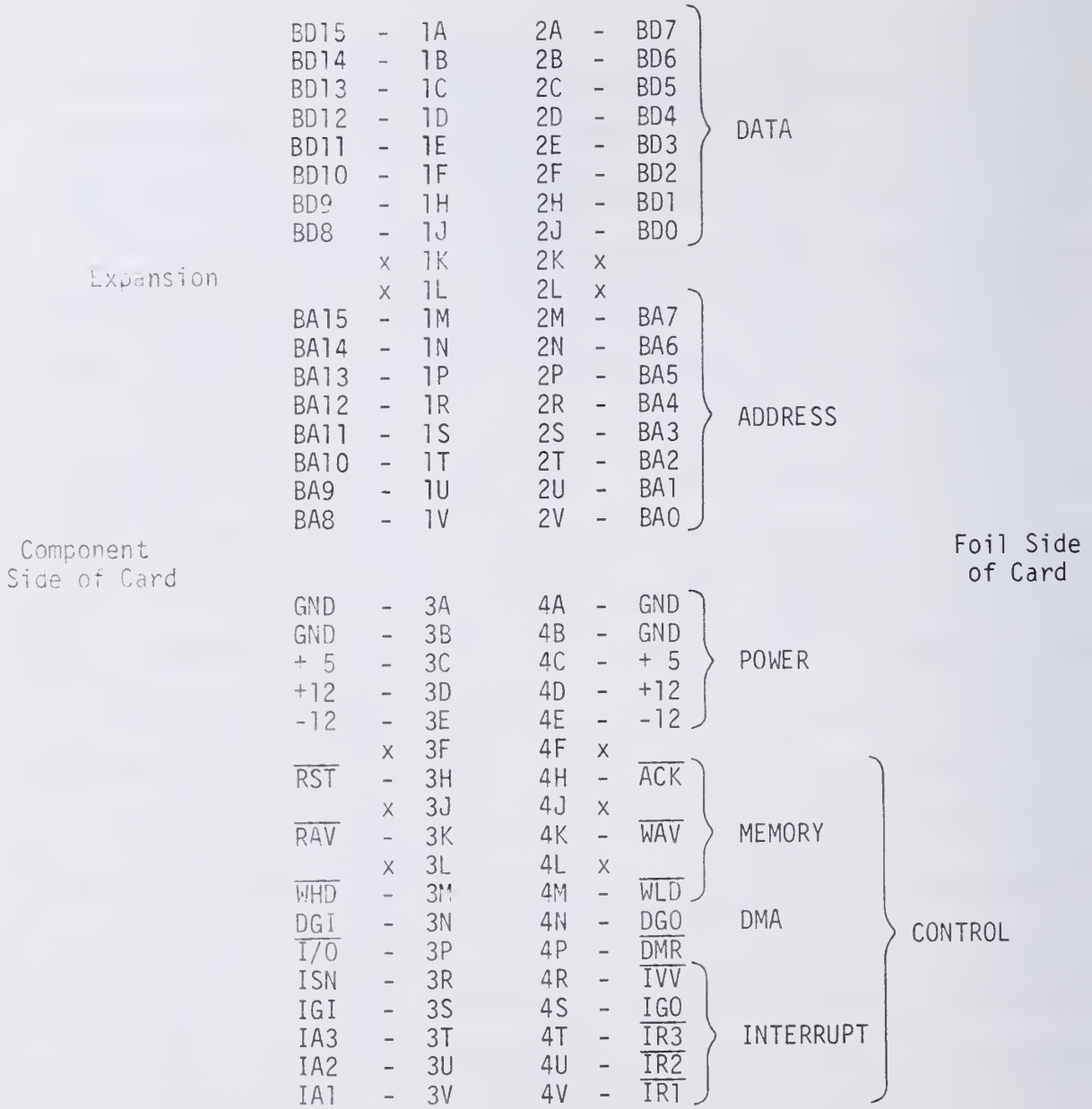


Figure 7. Backplane signal layout (pin side view).

The 16 bus data lines are the top pins of a slot (cards are inserted so that they stand upright). The data lines are split into two groups of eight, one on each side of the slot. Thus, when the data lines are "wiggled" through to the next slot, data lines are not subject to cross-talk from another group of signals. The same holds true for the bus address lines. The four uncommitted lines between the data and address groups are for possible expansion of either (or both) groups.

The DEC H803 connector block is meant to be mounted on a single bar that fits into a 19" rack. The connectors straddle the bar and are held in place by screws. It is possible to use the bar as a ground bus and to mount a voltage distribution bus to the bar. The power supply pins were located next to the mounting bar for these reasons.

Control signals are grouped according to function for ease of location when troubleshooting. Also, there are six spare lines surrounding the memory control signals that may be grounded until such time as other purposes arise for them. Cross-talk will be decreased among the power lines and memory control signals in this way.

The daisy chain signals (DGI, DGO, IGI, IGO) feed straight across the connector so that cards not using them need only pass the signals through from the front to the back of the card. This also simplified the layout of the backplane circuit card. Since the DGI and IGI lines are on the left side (viewed from the connector's back) the CPU card which issues these signals must be in the leftmost slot. Viewed from the card side of the bus, the CPU is in the RIGHTMOST slot.

Another convention to which MUMS module designers should adhere to is putting components on the right side of the card (as it is viewed from the card side of the bus).

2.7 Bus Termination

To reduce signal reflections on the bus, resistors to +5 volts and ground are placed at each physical end of the bus. These resistors also perform a pull-up function for the open collector control signals. The matching network is shown in Figure 8. The values of the resistors were chosen to guarantee a logic 1 and also attempt to match the characteristic impedance of the bus. The network must be on every signal line except the daisy chains. To make construction of the bus easier, an etched termination board was produced that can be slipped over the pins of a connector and soldered in place.

Termination of the bus in this manner does, however, reduce the fan-out from gates that drive the bus. For this reason, it is suggested that low input current buffers be used when bringing a bus signal onto a card. Typical buffers used in the MUMS system are Signetics 8T97s, low power Schottky gates, and Intel 8216 bi-directional bus drivers.

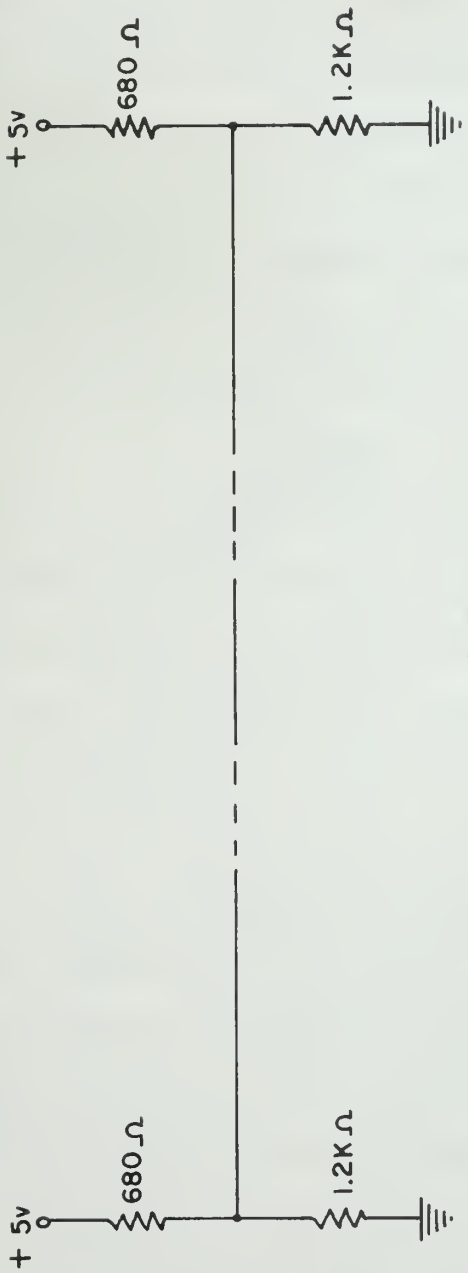


Figure 8. Termination network.

3. MICROCOMPUTER INTERFACES

3.1 Intel 8080 MUMS CPU Module

The Intel 8080 has become one of the most available and cheapest of single chip microcomputers. An interface between the 8 bit 8080 CPU and the MUMS bus protocol will be described here. Little or no attempt will be made to explain in detail or justify the exact timing of the 8080. Such detailed information should be obtained from the manufacturer's data pamphlet [2].

The 8080 interface is shown in Figure 9. In addition to the 8080, only 25 ICs are needed for buffering, generation of the MUMS control signals, priority interrupt control, and necessary clocks. This circuitry fits well into the "dual card" format chosen for MUMS.

Some signals that the 8080 generates need only be buffered before being tied to the MUMS bus. Hold Acknowledge (HLDA), "HOLD", "READY", and RESET can be connected to the bus with minimal preparation. $\overline{\text{DMR}}$ and $\overline{\text{ACK}}$ must be synchronized to 8080 clocks to guarantee set-up times. Power-up reset is generated by an RC network feeding the MUMS bus side of the RESET line.

Address lines must not only be buffered, but also be tri-state to support the DMA feature. Bi-directional tri-state buffers are used for the data lines. Although, whenever the direction of data flow between the MUMS bus and 8080 disagree, both sides of the data buffer must be disabled. Direction of data flow from the 8080 is determined with the DBIN signal. Whenever a Read or Input cycle is being executed, the data driver's chip select is disabled until DBIN goes active, thus resolving any conflict.

Information regarding the type of machine cycle being executed is latched up from the 8080 data lines during certain clock times. With very

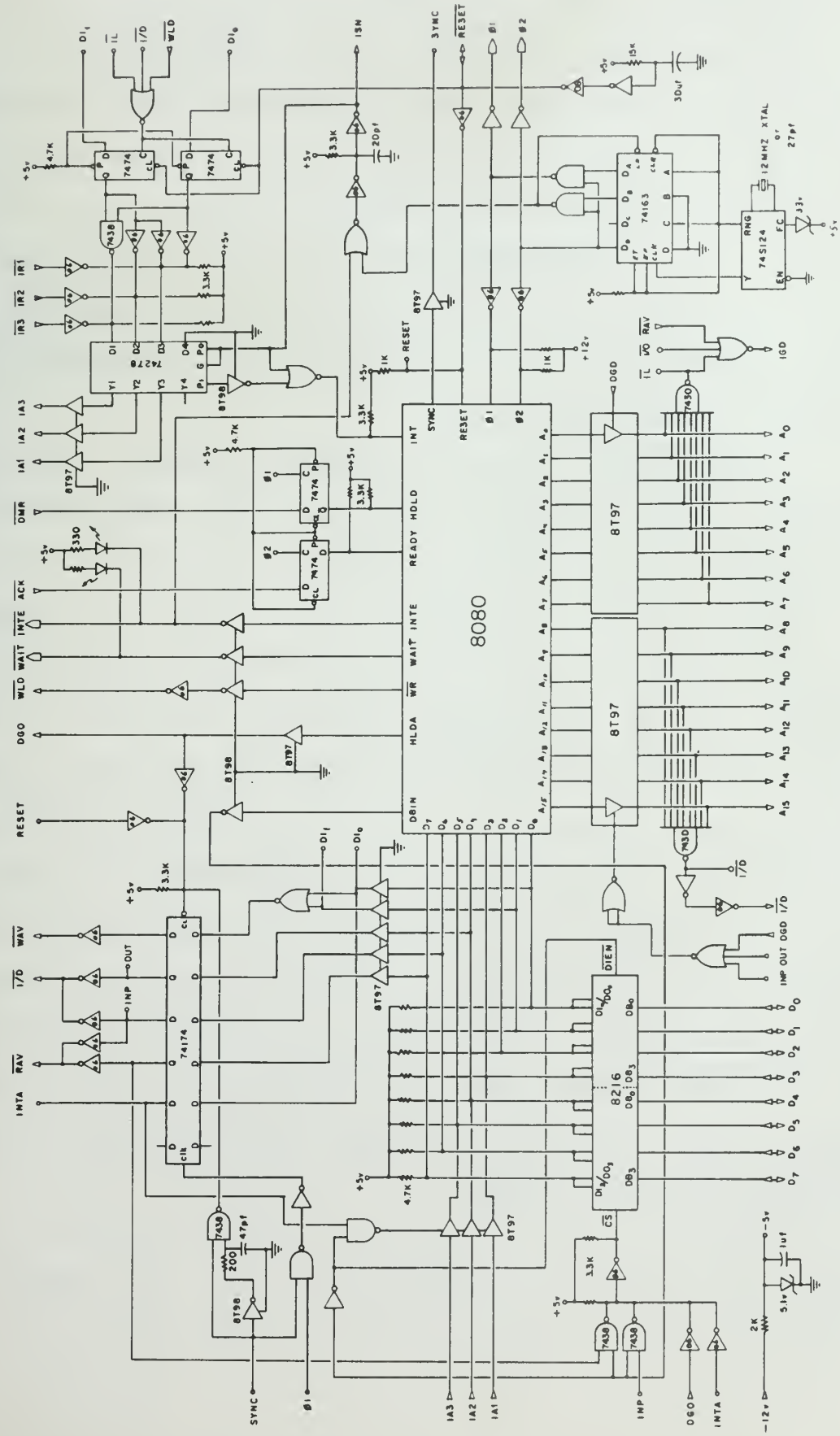


Figure 9. Intel 8080 MUMS CPU interface.

little decoding, the status data becomes $\overline{RAV}$, $\overline{WAV}$, and $\overline{I/O}$. The $\overline{I/O}$ signal, when decoded from status, means that an INPUT or OUTPUT instruction is being executed (not memory mapped). However, the 8080 duplicates the port address on the upper and lower 8 bits of the address bus. To prevent memory modules from responding, the upper 8 bits of buffer are disabled. Since the bus floats high, and no memory module resides on the "I/O page" (all ones), the conflict is resolved. If a memory referencing instruction addresses the I/O page (memory mapped I/O), an 8 input NAND gate will respond by asserting $\overline{I/O}$ through an open collector buffer.

The most complex section of the 8080/MUMS interface is the interrupt controller. One cannot easily understand its function without first looking at how the 8080 enters an interrupt sequence.

Whenever the internal interrupt enable flip-flop is set, the 8080 will honor a request appearing at the INT pin. After the current instruction is completed, an interrupt acknowledge condition will be output as the machine status (latched up from the data lines). At this time, the 8080 expects to see an instruction gated onto the data lines by the interrupt controller. This is usually a $RESTART_n$ instruction which is, in effect, a one byte CALL statement. This instruction causes a subroutine jump to be performed to one of eight different locations depending upon the value of n. A rudimentary form of interrupt vectoring is implemented in this way.

The MUMS interrupt controller has the ability to disable different levels of requests through the use of output port number 377_8. The low order 2 bits are coded as follows:

BIT_1	BIT_0	MEANING
0	0	ENABLE ALL LEVELS
0	1	ENABLE LEVELS 2 AND 3
1	0	ENABLE LEVEL 3 ONLY
1	1	ALL LEVELS DISABLED

The output port consists of an 8 bit decoder on the low order address lines and 2 bits of data latch. The outputs of the latch are decoded with open collector gates which permit interrupt requests to be recognized when they are in the high state. The highest priority pending interrupt is selected by the 74278 priority latch and passed to the appropriate acknowledge line. If any request is present, it will be latched and presented to the 8080 when ISN goes low. ISN is generated by the 8080 whenever the internal interrupt enable is set. Since this internal flip-flop is reset automatically upon interrupt acknowledge, ISN will not be sent out again until an ENABLE INTERRUPT instruction is executed.

The RESTART instruction mentioned earlier has the bit pattern 11AAA111 (where AAA defines the vector address). The one Interrupt Acknowledge signal that is asserted by the priority latch is gated to the data bus during the 8080 interrupt acknowledge time. In this manner, one of three different RESTART instructions will be executed depending upon the interrupt level selected by the priority latch.

Of course, some necessary software is also part of the interrupt system. The IGO signal is generated whenever an input (or read) is done from port number 377_8. This input is employed to identify the interrupting device for the service routine, and also to reset the request latch on the

interrupting device. This identification vector can be used as an offset into a table of service routine addresses to provide a physical to logical unit mapping.

3.2 Other CPU Interfaces

The general aspects of the 8080 interface just described can be applied to most other microcomputers that may be mated to the MUMS system. The general techniques for address and data buffering may be applied as well as specific parts of the interrupt handling circuit. As an example of what sort of problems might arise in the design of a new CPU interface consider the RCA COSMAC[1]. The input and output instruction of the COSMAC cannot be supported by MUMS because of the method of data transfer. I/O is not through a CPU register but instead, the CPU identifies both a memory location and an I/O port number. The data transfer is directed between the memory and the I/O port by (and not through) the CPU.

Another example of an interface problem is a physical one. The DEC LSI-11 is available as a card only, containing an on-board dynamic RAM. An interface from MUMS to an LSI-11 must not only match the processor's control signal convention (which is rather similar to that of MUMS), but must also allow for DMA to both MUMS memories and the LSI-11's RAM.

The MUMS bus protocol was designed to be asynchronous and speed independent. But of course, the propagation delays found in the implemented modules and the bus capacitance will limit the actual speed of operation. For instance, the address recognition logic on a 4K x 8 RAM module will require approximately 75ns to generate the $\overline{ACK}$ signal (not including bus

---

[1] This microprocessor has since been renamed the RCA CDP 1801.

delays). Obviously, this must be taken into account when designing an interface for a Schottky bit slice processor.

4. OTHER SYSTEM MODULES

4.1 4K x 8 RAM Module

A 4K x 8 Random Access Memory module has been fabricated using the 2102 type 1K semiconductor RAM. Figure 10 shows the logic design of this MUMS module. The page address of the module may be set with four switches. The positions of these switches are compared to the high order address bits by open collector exclusive-or gates. Equality will enable the bi-directional data drivers as well as the read cycle one-shot timer. $\overline{RAV}$ will fire this timer and define the direction of data flow through the buffers. $\overline{ACK}$ will be asserted only when the module is addressed and is in the process of accessing or writing data. All address lines are buffered before being presented to the array of 32 memory chips. The chip select lines to one of the four 1K x 8 columns are driven from one half of a 74S139 decoder (the other half is employed as an OR gate!).

The module only stores an 8 bit wide word, but it is possible to jumper the data buffers to either the lower or upper 8 bits of the data bus. By having two modules occupy the same address space and different data byte positions, 16 bit wide words can be stored. Also selectable by switches is either the $\overline{WLD}$ or $\overline{WHD}$ signal. Of course, one should be careful not to connect both signals on a particular card.

The access time of the MUMS 4K x 8 RAM module cannot be specified here since it depends entirely upon which particular version of 2102 memory is used. Access times could be anywhere from 250ns to 1.5μs for a 2102 device. In figure 10, on the 74123 dual one-shot, the 150pf capacitor allows for a read access time of about 600ns, while the 39pf capacitor is for a write pulse width of about 200ns. These component values should be adjusted to suit the characteristics of the chosen memory chips.

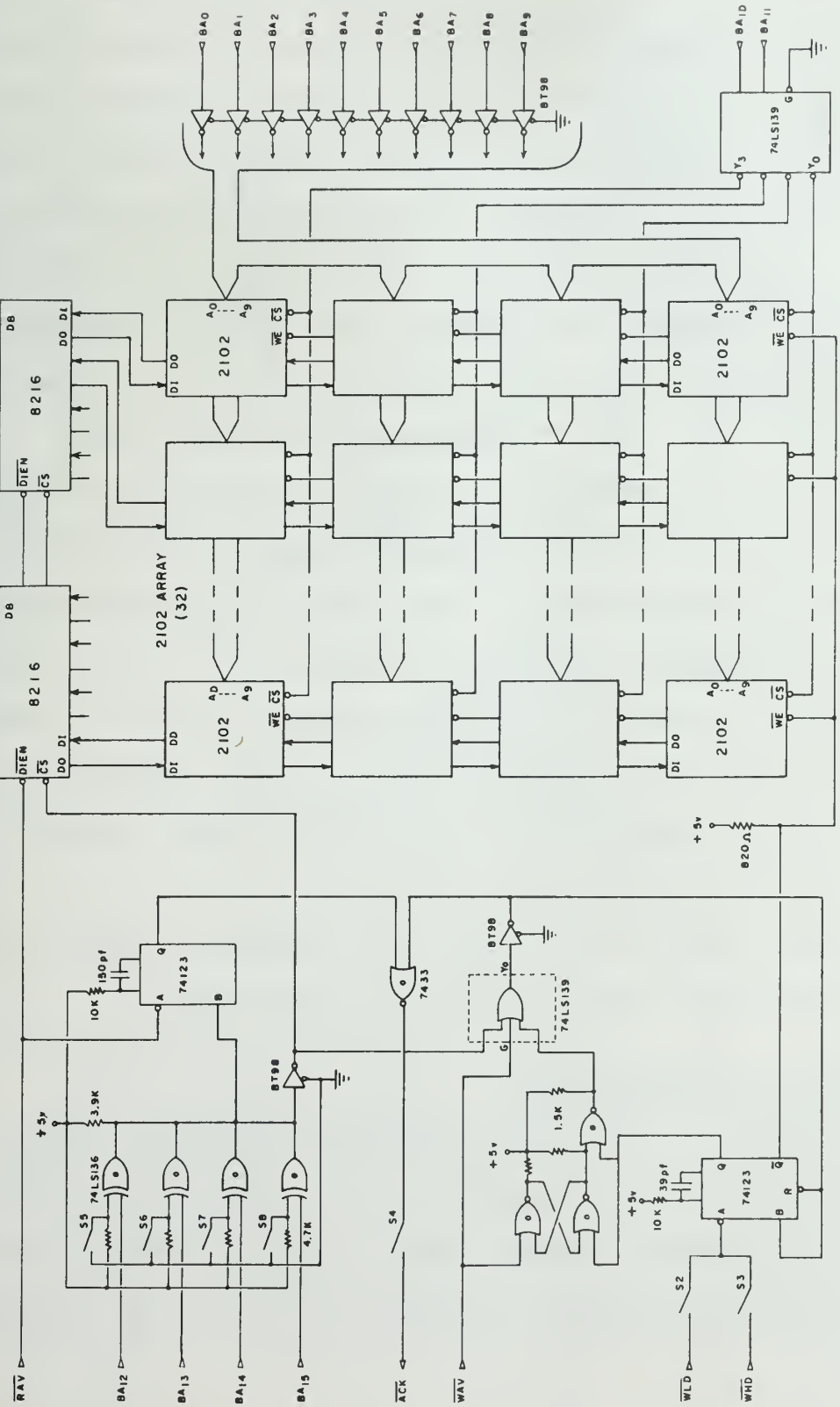


Figure 10. MUMS 4K X 8 RAM module.

4.2 Read-Only Memory Module

The design of a 4K x 8 Read-Only Memory module is shown in figure 11. Either the National MM5203 or the 1702 type of erasable ROM may be used on the module to provide non-volatile memory in increments of 256 bytes. Address lines to the ROM array are buffered by Signetics 8T97 non-inverting gates. These gates are also tri-state and are used to connect to the MUMS data bus as well. The 16 necessary chip select lines are decoded from address bits 8 through 11 by a 74L154. The high order four address bits are compared to a switch register so that the card may be placed in any of 16 different pages in the address space.

The array of ROMs must all be of the same type (either MM5203 or 1702) because different supply voltages are needed. The 1702 needs -9 volts which is derived from the -12 volt pins on the bus. If MM5203 chips are used, the regulator is removed and a jumper put in its place to provide the necessary -12 volts. There is also a switch on the card for changing pin 13 of the ROMs from ground to +5 volts. The 1702 requires that pin 13 (program) be +5 volts during normal operation, whereas the MM5203 requires that it be ground (A_9).

Again, the eight data outputs may be connected to either the upper or lower byte of the data bus to provide for 16 bit words.

4.3 Serial I/O Interface Module

The serial I/O module allows the use of any standard RS-232 or 20ma current loop device with the MUMS system. These devices include teletypes, CRT terminals, acoustic couplers, etc. The module also provides an interface to a standard audio cassette recorder which can be used as an

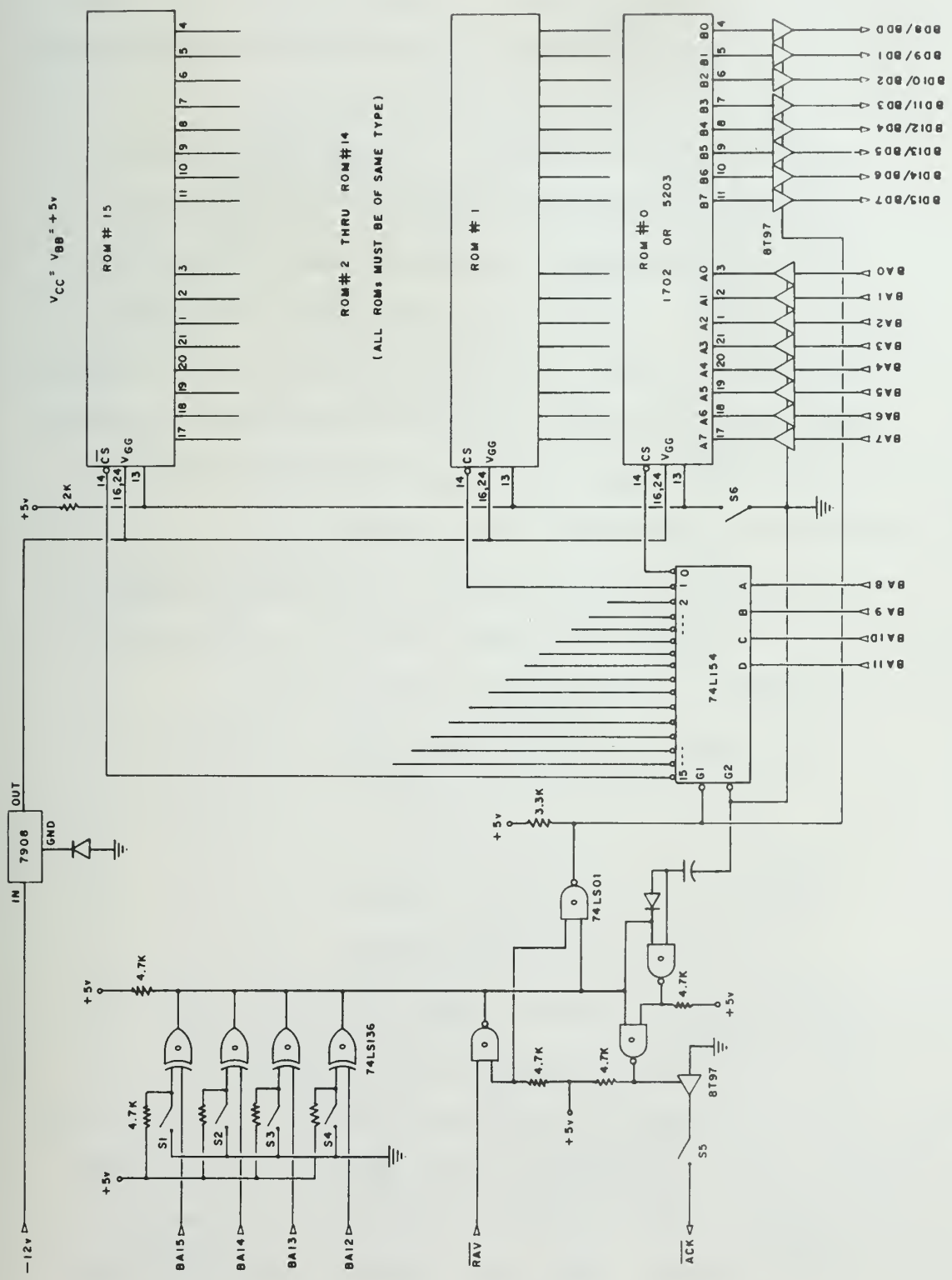


Figure 11. Read-only memory module.

inexpensive program and data storage device. One of two predetermined baud rates may be selected by switches on the module. Also switch selectable are the number of data bits and stop bits per word, even/odd/no parity, and the addresses of the I/O ports on the card.

The heart of the serial I/O module is the UART chip (Universal Asynchronous Receiver/Transmitter). It performs all the functions of serializing words to be transmitted, including the addition of parity, stop and start bits. Also, it puts a serial input stream into parallel form and checks for parity, buffer over-run, and framing errors. A specific example of a UART that may be used on the module is the General Instrument AY-5-1013 (although there are many other equivalent devices). The schematic for the serial I/O module is shown in figure 12.

The module has four I/O ports defined by the low order three bits of the address bus. The upper five I/O port address bits are selected with a switch register. Assignments for the four ports are:

PORT #	USE
0	Receiver Status
2	Receiver Buffer
4	Transmitter Status
6	Transmitter Buffer

The use of these I/O ports is patterned after the PDP-11 where bit 7 of status is "DONE", and bit 6 is the internal Interrupt Enable. Also, the actual port number assignments for the low order address bits are the same as for the PDP-11. Thus, instructing students in the programming of the module should be easier and more relevant. In addition to the receiver status bits mentioned already, parity error, framing, and over-run errors are indicated by bits 3, 2, and 1 respectively.

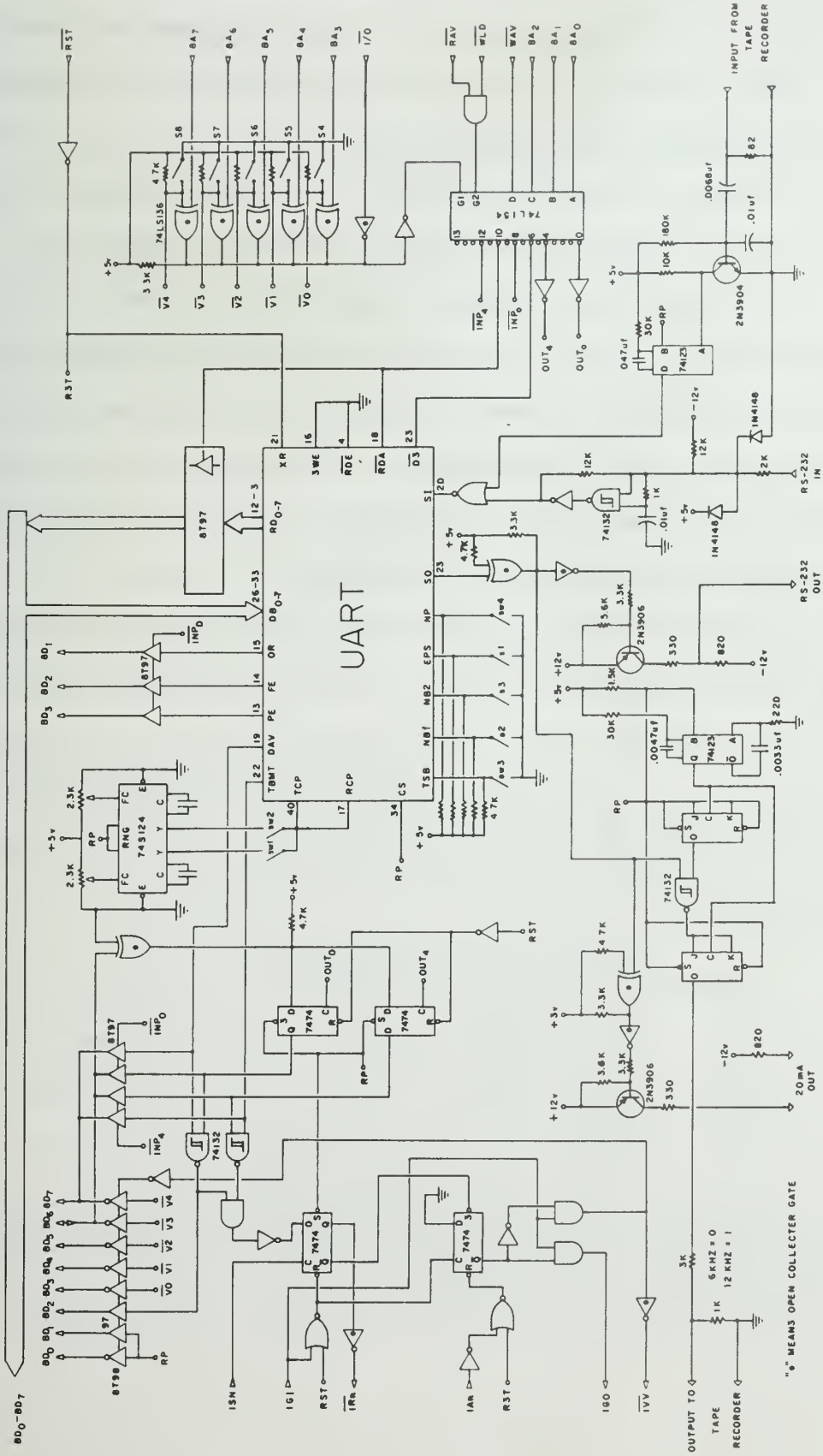


Figure 12. Serial I/O interface module.

The interrupt request logic is virtually the same as in figure 4 except that the system reset signal is also applied to clear the request flip-flops. The vector that is gated onto the data bus when IGI is received reflects the port numbers of the particular card and interrupting condition. That is, if switches S4 through S8 are all zero, a "done" interrupt issued by the transmitter will have a vector of 006 (transmitter buffer port number). Conversely, a "done" interrupt issued by the receiver would have the vector 002. These vectors can be used as the low order address into a page of memory that contains service routine addresses. Or, in simpler systems, the vector could be interpreted as representing a certain class of device if it is in some particular arithmetic range (e.g. all teletype interfaces might have vectors greater than 300_8).

The baud rate at which the UART will operate is governed by the 74S124 dual VCO. The baud rate will be 1/16 the frequency of the VCO that is selected using SW1 or SW2. The number of stop bits to be transmitted is controlled by SW3 (a logic "one" will send two stop bits and only one if a logic "zero"). The number of bits per character is selected by S2 and S3 as follows:

S3	S2	Bits/Character
0	0	5
0	1	6
1	0	7
1	1	8

Parity will be transmitted if SW4 is a logic "zero" and S1 controls even or odd parity (even parity if logic "one").

The serial output section has three parts; the RS-232 compatible output, 20ma current loop output, and frequency shift keying for cassette output. Their logic polarities are:

	Logic Zero	Logic One
EIA RS-232	+5 volts	-12 volts
20ma current loop	0ma	20ma
FSK	6 KHZ	12 KHZ

It should be noted here that the FSK output is intended to be for 300 baud or less. The modulation frequency of 6 KHZ is as high as can be reliably reproduced on an inexpensive audio cassette deck. The 12 KHZ tone is only used to fool the automatic record level circuitry found in many cassette decks. The 12 KHZ will be sensed by the electronics but not recorded on the tape. Thus, the differing bit densities found in the serial output stream will not change the setting of the automatic record level circuit.

4.4 Shared Memory Display Controller

An alphanumeric display controller module has been built that can provide low cost and very high speed output to the MUMS user. It consists of a 1K x 8 block of RAM that is normally under the control of refresh logic, but switches over to be controlled by the CPU whenever an "output" or "screen read" is performed. The CPU sees nothing but normal random access memory, yet the user is able to see everything the processor writes into the RAM as characters on a TV screen. The controller is capable of displaying 16 lines of either 32 or 64 characters per line. Characters are stored in the RAM by the CPU in 7 bit ASCII code with the lowest address corresponding to the upper left character position. As the address

is incremented, the associated character position moves from left to right and top to bottom. The eighth bit of each character position is used to indicate the presence of a cursor - either a block, or a line under the character. The cursors may also be made to flash. These options for the cursors, as well as the number of characters per line, black on white display or vice versa, page select (when in narrow display mode), and scrolling up or down are all selectable through an output port.

The complete alphanumeric display controller is shown in figures 13, 14, and 15. Two clocks are used in the circuit to provide a time base for the TV sync generator and a dot rate signal to serialize the character generator output. The horizontal and vertical sync pulses from the TV sync IC are used to initiate the sequence of events for a scan of the TV screen. The dot rate oscillator is disabled for a time after V-sync by one-shot "I" to provide a top margin. One-shot "H" delays after H-sync to provide a left margin. Right and bottom margins are sensed after the correct number of characters have been counted and latched into flip-flops "S", "K", and "L". Every eighth dot oscillator pulse will generate an "increment character" signal used to bump the refresh address, latch the current memory output, and to latch the character generator output into the output shift register. The current text row is determined by counters "B" and "C". Text column is determined by counters "D" and "E", with "F" providing a changeable offset at V-sync for scrolling.

The refresh memory address selection is controlled by the CPU alone. As long as the screen is not being continuously read or written by the CPU, the user will probably not notice the interruptions in character refreshing.

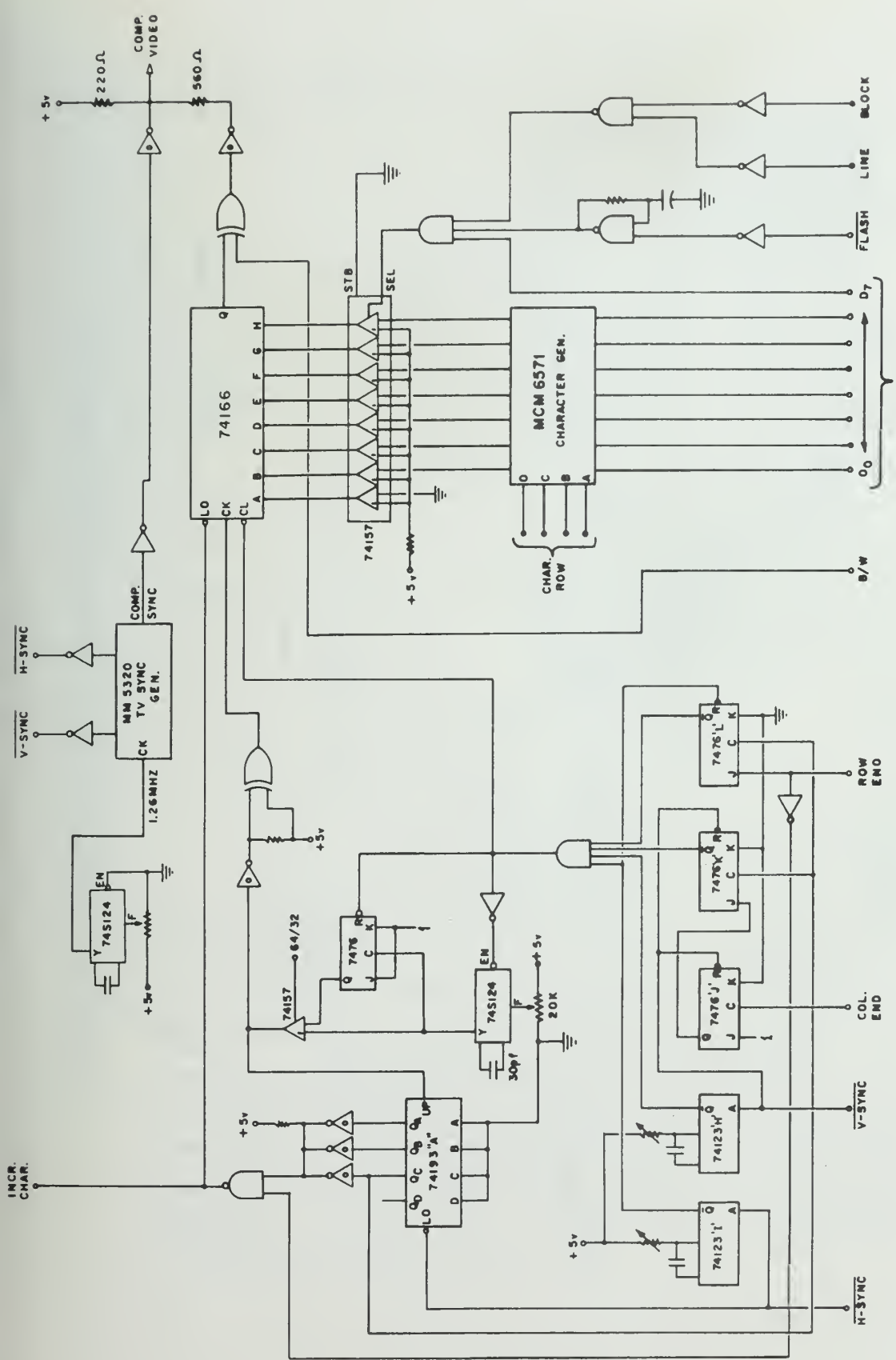


Figure 13. Display Controller (Character Generation).

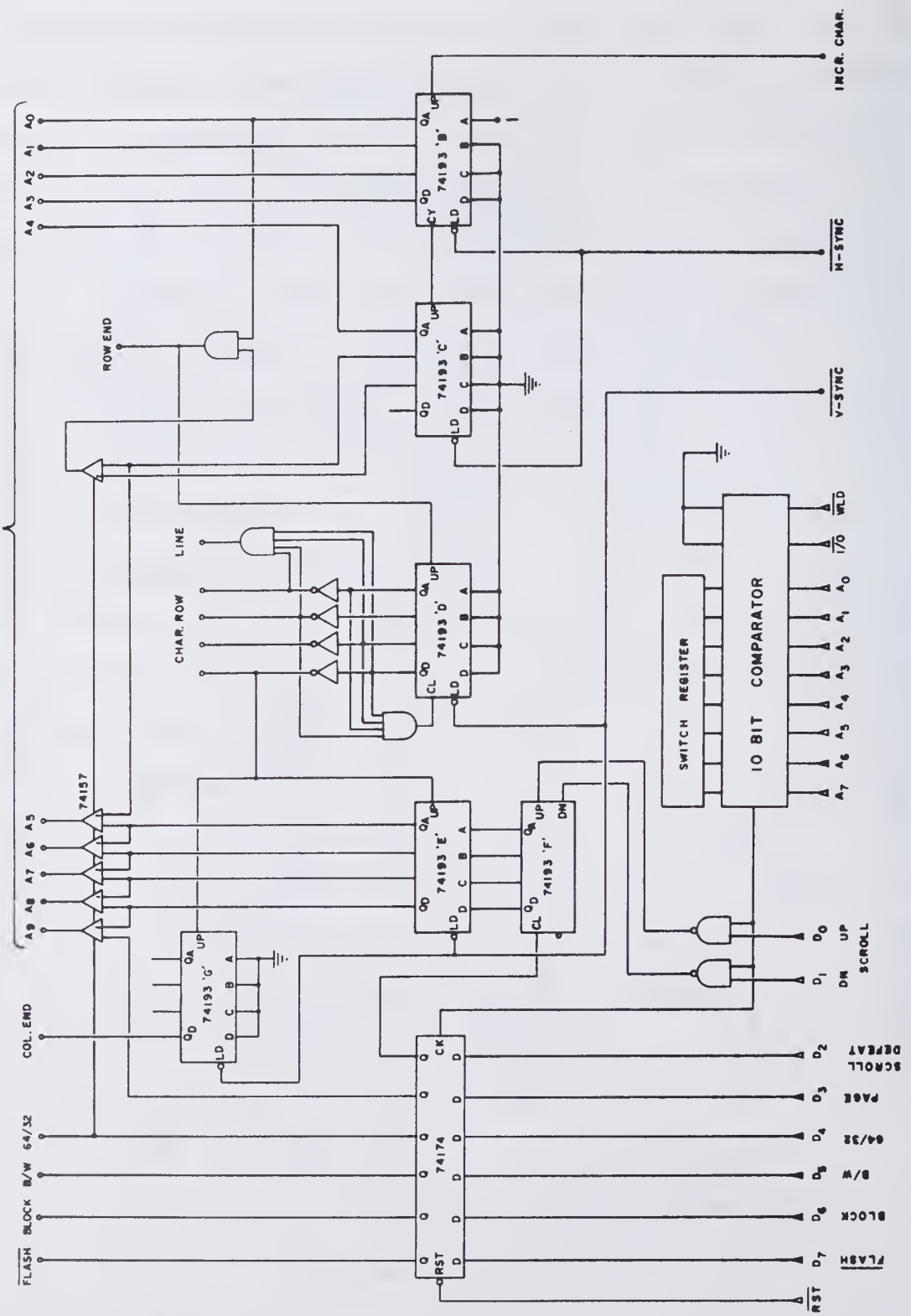


Figure 14. Display Controller (Refresh Address Generation).

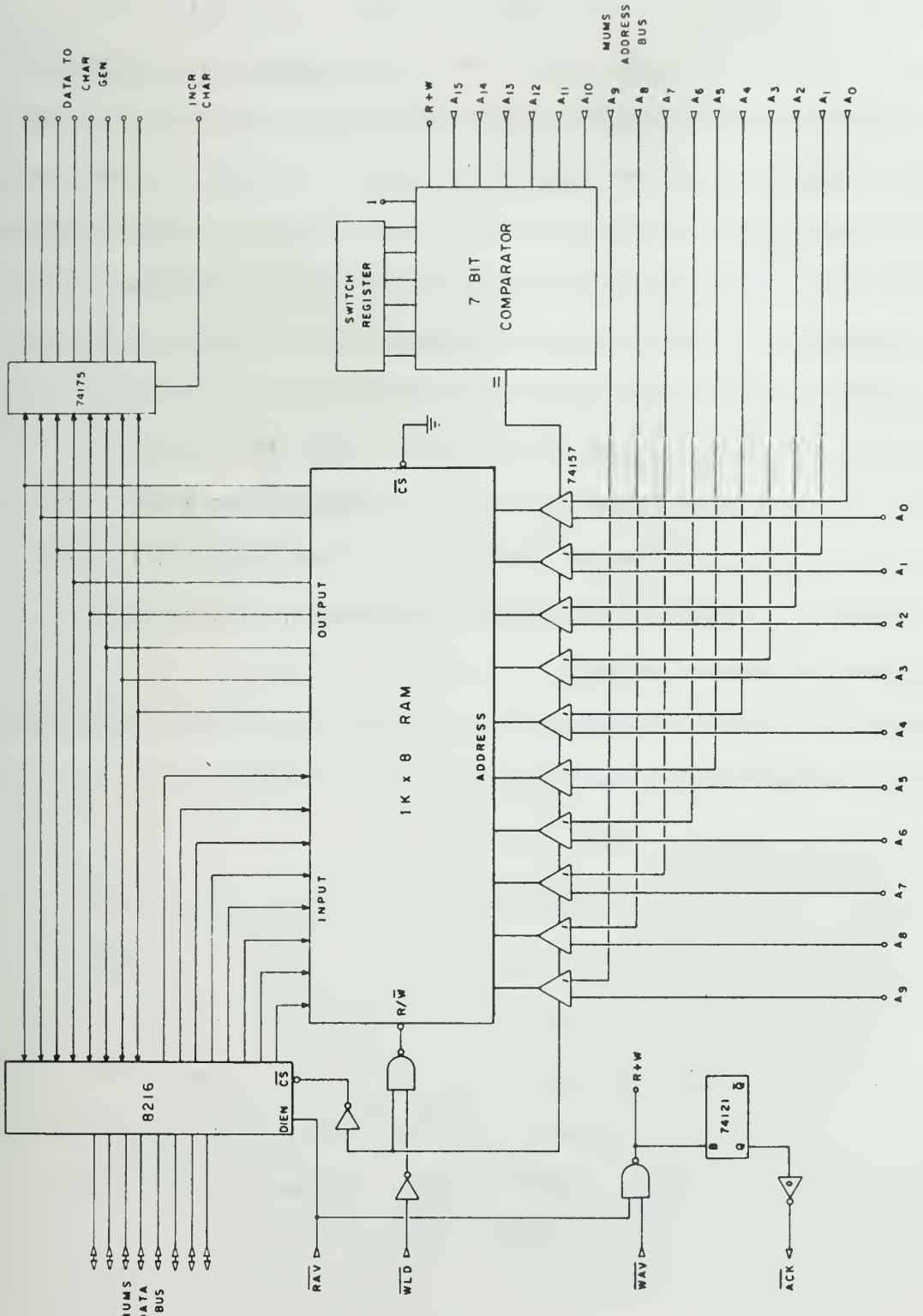


Figure 15. Display Controller (Refresh Memory).

Of course, some software is needed to perform the traditionally hardware tasks of screen erasing, cursor control, and recognition of control characters that would be sent to any conventional display. But by using the CPU for these functions, a higher degree of display intelligence is realized.

In addition to this alphanumeric display controller, a random vector display controller was designed and built for MUMS II by Robert Pleva. This vector interpolator is a DMA device that refreshes an X-Y CRT from a display buffer prepared by the CPU. The display buffer, which resides in main memory, consists of triples of vector information. Each triple contains X and Y screen position, and intensity for the next line to be drawn. The controller picks up triples from the display buffer through DMA until the end of the buffer is reached. The buffer is continuously displayed from beginning to end to refresh the CRT.

A complete discussion of the design and operation of this controller is available as a Department of Computer Science report [3].

4.5 Control/Display Module

Although the control of the MUMS system is intended to be done through a programmed "front panel", certain hardware debug aids should be provided. A simple module that latches address and data for display by LEDs, and has a manual reset and "single step" feature has proven to be invaluable. The "single step" feature can be implemented quite easily by asserting the $\overline{\text{ACK}}$ line whenever a "WAIT" switch is thrown. A "STEP" button releases the $\overline{\text{ACK}}$ line until the next $\overline{\text{WAV}}$ or $\overline{\text{RAV}}$ leading edge.

5. MUMS SYSTEM SOFTWARE

5.1 8080 Processor Initialization

The 8080 processor module requires that certain registers and flags be initialized upon reset. The stack pointer and interrupt enable flag within the CPU must be set as well as the interrupt level enable register outside of the CPU. These tasks are performed by instructions starting at location zero in memory. Upon receipt of the $\overline{RST}$ signal, the 8080 will commence execution at location zero. Obviously, these instructions must be available as soon as power is turned on. Therefore, the lowest 4K of memory should be a ROM module containing the necessary routines.

Part of the MUMS interrupt scheme for the 8080 module is supported in firmware. The IG signal propagated down the bus is actually generated by an input instruction to port number 377_8. The data received from this input operation is the interrupt vector from a requesting device. The scheme to enter an interrupt sequence (described in section 3.1) involves the execution of a RESTART instruction. Either a RST_1, RST_2, or RST_4 instruction will be jammed by the hardware during an interrupt acknowledge sequence. Being automatically sent to one of three different locations depending upon the level to be serviced will allow the disabling of all but higher levels. In this way, the servicing of a low priority interrupt may itself be interrupted by a higher priority device.

A possible implementation of the initialization, interrupt support, and bootstrap routines for the 8080 module is shown in Appendix A. Although the code shown has not been tested, it does illustrate how the routines fit into the operation of the MUMS system.

Reset is accomplished by the routine called RSTØ. This routine first sets the stack pointer and then loops indefinitely enabling interrupts for the bootstrap serial I/O module. It is assumed that a RAM module will reside in the second 4K slot from the top memory address (the I/O page is in the top slot) so the stack can grow downwards indefinitely.

Nesting of interrupt requests of different levels is accomplished by the routines called RST1, RST2, and RST4. Each routine will disable all but higher interrupt levels and then join at a common section. The interrupt vector will be acquired and used as an index into a page of ROM that contains service routine addresses. This page of addresses is assumed to be the second page from location zero (that is, addresses 0100_{16} through $01FF_{16}$). Since addresses are two bytes each, this firmware will support 128 device vectors at even addresses in the index page.

The address contained in the first two bytes of the index page is that of a routine called BOOT. This routine will be entered whenever an interrupt is received from a serial I/O module with the zero vector address. Programs and data may be loaded and executed through the use of this serial interface and a teletype or cassette deck. The BOOT routine will input a character from the serial interface and interpret it as either a hex digit or a command. If it is a hex digit, it will be OR'ed into the low order 4 bits of a 16 bit value ("DATA"). If the character is a "#", the "DATA" variable will be loaded into "ADDR". If it is a ".", then the low order 8 bits of "DATA" will be deposited into the memory location addressed by "ADDR" (which is then incremented). And if the character is an "X", execution will start at the location addressed by "ADDR". Thus, the string

0544#C3.44.05.0544#X will load and execute a program at 0544_{16} (a JMP instruction that jumps to itself).

5.2 Universal DEBUG Monitor and Cross-Assembler

To enhance the interchangeability of MUMS processing elements, a common "DEBUG" routine should be implemented for each CPU. This DEBUG program would allow the user to set break points in a program or inspect and alter memory locations in either decimal, hex, or octal bases. If a common command set and usage for such a program were defined, a user of different processing elements would only have to learn a single system program. Commands should be as mnemonic as possible and still be only one character. This would make for a simpler interrupt driven program. This DEBUG program would be merely a more sophisticated version of the "BOOT" routine already described.

Another important capability is that of assembling machine code from symbolic instructions. Instead of writing a new resident assembler for each processor, a Universal Cross-Assembler has been developed by Charles Kominczak. This cross-assembler is described fully in his thesis issued as a Department of Computer Science report [4]. Briefly, the cross-assembler is written in PASCAL and runs on either a DEC SYSTEM 10 or PDP-11. It is table driven and can assemble programs written for an INTEL 8080, MOS Technology 6502, RCA COSMAC, Fairchild F8, or a National IMP-16. Its output is in a hex loader format that allows both a reasonably compact file and the capability of backing up only a short distance if a checksum error is found during loading.

6. CONCLUSION

The set of 18 MUMS control signals in addition to the 32 data and address lines constitute a very small set of generic signals. This bus protocol is simple enough to grasp quickly, yet offers great versatility and expandability. It is asynchronous and speed independent (up to the limits of propagation delay). The bus may also be constructed in increments of four card slots to allow for systems that are only as large as are needed. Any microprocessor should be easily interfaced to the MUMS protocol which supports DMA and maskable interrupt capabilities.

The design of the MUMS III system is only the beginning. There are many other tasks and projects related to MUMS that are underway. Other CPU interfaces are being implemented to support the many microprocessors that have so graciously been donated to us by their manufacturers. Also, a floppy disk controller is being designed as well as a communication unit that will allow expensive peripherals to be shared. This communication unit should also support a multiprocessing scheme for multiple MUMS busses. Although no plans have been made for its construction, a "student proof" chassis and power supply would also be a reasonable pursuit.

In addition to the Universal Cross-Assembler, work on the definition for a universal assembly language is progressing that will make transferable MUMS software possible. This universal assembly language (called M-code) will consist of the most machine independent assembly level concepts and features that can be defined. Compilers that generate M-code, and resident M-code assemblers/interpreters for each processor will allow both transferal of compiler output and processor interchangeability.

It is hoped that the MUMS system will provide a versatile, easy to use, and lasting tool for research and the teaching of Computer Science.

REFERENCES

[1] IEEE Standard Modular Instrumentation and Digital Interface System (CAMAC), The Institute of Electrical and Electronics Engineers, Inc., New York, 1975.

[2] 8080 Microcomputer System User's Manual, INTEL Corporation, Santa Clara, CA, 1975.

[3] Pleva, Robert M., "Low-Cost Vector Graphics for a Microprocessor System," Department of Computer Science report UIUCDCS-R-76-808, University of Illinois at Urbana-Champaign, Urbana, IL, July, 1976.

[4] Kominczak, Charles P., "A Universal Cross-Assembler," Department of Computer Science Report UIUCDCS-R-76-803, University of Illinois at Urbana-Champaign, Urbana, IL, June, 1976.

APPENDIX A

8080 SYSTEM FIRMWARE

```
          ORG       ØØØØH
RSTØ:     LXI   SP  EFF9H     ;   LOAD STACK POINTER
          JMP       EILOOP

          ORG       ØØØ8H
RST1:     PUSH      PSW       ;   LOW LEVEL INTERRUPT
          MVI   A   Ø1
          JMP       VECT

          ORG       ØØ1ØH
RST2:     PUSH      PSW       ;   MED LEVEL INTERRUPT
          MVI   A   Ø2
          JMP       VECT

          ORG       ØØ2ØH
RST4:     PUSH      PSW       ;   HIGH LEVEL INTERRUPT
          MVI   A   Ø3
          JMP       VECT

          ORG       ØØ4ØH
EILOOP:   XRA   A
          STA       EICODE
          OUT       FFH
          DCR   A
          OUT       ØØ        ;   DAV INTE ON SERIAL CARD # Ø
          EI
          JMP       EILOOP
;
;    VECTORED JUMP TO INTERRUPT SERVICE ROUTINE
;
VECT:     PUSH      B
          PUSH      D
          PUSH      H
          MOV       B, A      ;   COPY EI STATUS TO STACK
          LDA       EICODE
          PUSH      PSW
          MOV       A, B
          OUT       FFH
          STA       EICODE
          MVI   H   Ø1        ;   GET VECTOR FROM DEVICE
          INP       FFH
          MOV       L, A
          MOV       E, M
          INX       H
          MOV       D, M
          XCHG
          EI
          PCHL                ;   JUMP TO SERVICE ROUTINE
```

```
;
;    RETURN FROM SERVICE ROUTINE
;
RTS:      DI
          POP       PSW       ;    GET OLD EI STATUS
          STA       EICODE
          OUT       FFH
          POP       H
          POP       D
          POP       B
          POP       PSW
          EI
          RET
EICODE:   EQU       EFFFH
;
;    BOOTSTRAP LOADER
;
BOOT:     INP       Ø2        ;    SERIAL INTERFACE FOR BOOT
          CPI       'Ø'
          JC        TESTN
          CPI       '.'
          JC        PUSHD     ;    OK TO PUSH DATA
          CPI       'A'
          JC        RTS
          CPI       'G'
          JC        PUSHD     ;    OK TO PUSH DATA
          CPI       'X'
          JNZ       RTS
          LHLD      DATA      ;    EXECUTE
          XCHG
          LHLD      ADDR
          DI
          LXI   SP  EFF9H
          XRA   A
          STA       EICODE
          OUT       FFH
          EI
          PCHL                ;    JUMP TO (ADDR)
TESTN:    CPI       '.'
          JZ        LDATA
          CPI       '#'
          JNZ       RTS       ;    NOT VALID CHARACTER
          LHLD      DATA
          SHLD      ADDR      ;    LOAD NEW "ADDR"
          JMP       RTS
LDATA:    LDA       DATA
          LHLD      ADDR
          MOV       M, A      ;    LOAD DATA TO MEMORY
          INX       H
          SHLD      ADDR
          JMP       RTS
```

```
PUSHD:  LHLD    DATA    ;   PUSH HEX DATA
        DAD     H
        DAD     H       ;   MAKE ROOM ON LOW END OF WORD
        DAD     H
        DAD     H
        ANI     ØFH
        ORA     L
        MOV     L, A
        SHLD    DATA
        JMP     RTS
DATA:   EQU     EFFBH
ADDR:   EQU     EFFDH
```

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-809	2.	3. Recipient's Accession No.

4. Title and Subtitle

MUMS: A MODULAR UNIFIED MICROPROCESSOR SYSTEM

5. Report Date: June 1976

6.

7. Author(s)

Robert Walter Catlin

8. Performing Organization Rept. No. UIUCDCS-R-76-809

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Address

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

MUMS (Modular Unified Microprocessor System) is a bus protocol that takes advantage of the common properties that all microprocessors must possess. A small set of generic control signals has been defined that can be easuly produced by any microprocessor. The set of control signals is small, yet is complete and versatile.

Modules for the MUMS system (including CPU, RAM, ROM, and serial I/O) are also presented.

17. Key Words and Document Analysis. 17a. Descriptors

Modular microprocessor system
Generic control signals
Bus protocol
MUMS

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited	UNCLASSIFIED	56
	20. Security Class (This Page) UNCLASSIFIED	22. Price

SEP 1 7 1976

AUG 2 9 1980

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.807-811(1976
Experimental and formal language design
3 0112 088402851